Introduction

The objective of this assignment was to learn how stack operations are done and to give us more practice with MIPS assembly language by learning to implement nested routines. In C, this is equivalent to nested for loops. The challenge for this lab, however, was to use stack pointers instead of storing information directly in registers. This lab was an introduction to push, pop, and register addressing, as well as more practice with memory addressing.

Solution Methodology

For this assignment, we wrote a main routine in which two 6×1 vectors were defined in a similar manner to how numbers_to_use: was implemented in lab 0, with some of the values being negative numbers. We configured the stack pointer and passed the parameters to a function called dot_product which returned the result to main, as well as the two vector average values. All of this was done with overflow checking in mind. (see Source Code). With the routine built, we attached and programmed the chipKit Pro MX4 board and ran the instructions. Initially, the program performed the mathematical functions and went into an infinite loop (see Figure 1). Below is an example of the pseudocode used to design this recursive operation.

```
int dot (int i)
{
 if (i < 6)
      vector1(i)*vector2(i);
 else sum(vector1(i)+vector2(i));
}
```

One of the reasons for using the stack is to avoid delay hazards which results in registers being called before they are finished being executed. One solution is to push all registers that must be preserved onto the stack, just as we did with the saved registers in lab 0. The caller pushes any argument registers ($a0–$a3) or temporary registers ($t0–$t9) that are needed after the call. The callee pushes the return address register $ra and any saved registers ($s0–$s7) used by the callee. The stack pointer $sp is adjusted to account for number of registers placed on the stack. Upon the return, the registers are restored from memory and the stack pointer is readjusted.

Figure 1

These values are consistent with the calculations as performed in MATLAB (Figure 2).

Figure 2

Source Code

```
/*****************************************************************************/
/*                                                                           */
/*       ECE 344L    -    Microprocessors    -    Spring 2020                 */
/*                                                                           */
/*       kirby_lab03.c  -  Digital I/O and Finite State Machine              */
/*                                                                           */
/*****************************************************************************/
/*                                                                           */
/*       Author: David Kirby                                                 */
/*                                                                           */
/*****************************************************************************/
/*                                                                           */
/*       File Description:                                                   */
/*           Implements a finite state machine using the LEDs and buttons on */
/*           the chipKIT MX7 board.                                          */
/*                                                                           */
/*****************************************************************************/
/*                                                                           */
```

```
/*      Revision History:                                                     */
/*          Original Source Code by: E.J. Nava, 9/23/18                       */
/*          Modified Code by: David Kirby, 01-Mar-2020                        */
/*                                                                            */
/******************************************************************************/

#include <plib.h>

/* ------------------------------------------------------------------------ */
/*                              Configuration Bits                          */
/* ------------------------------------------------------------------------ */

// Configure MX7 board for debugging
#pragma config ICESEL = ICS_PGx1

// SYSCLK = 80 MHz (8 MHz Crystal/ FPLLIDIV * FPLLMUL / FPLLODIV)
// Primary Osc w/PLL (XT+,HS+,EC+PLL)

#pragma config FPLLMUL = MUL_20, FPLLIDIV = DIV_2
#pragma config FPLLODIV = DIV_1
#pragma config POSCMOD = EC, FNOSC = PRIPLL, FPBDIV = DIV_8
#pragma config FSOSCEN = OFF            // Secondary oscillator enable
#define SYS_FREQ (80000000L)

// *** these are preconfigured on the MX4 Board for a clock frequency of 80MHz
// *** and a PBCLK value of 10MHz.

/* ------------------------------------------------------------------------ */
/*                             Forward Declarations                         */
/* ------------------------------------------------------------------------ */

void DeviceInit();
void DelayInit();
void DelayMs(int cms);
void DisplayInit(int coins);


/* ------------------------------------------------------------------------ */
/*                                  Definitions                             */
/* ------------------------------------------------------------------------ */

#define cntMsDelay  10000           //timer 1 delay for 1ms

/* ------------------------------------------------------------------------ */
/*                                     Main                                 */
/* ------------------------------------------------------------------------ */

int main()
{
        int button_in12 = 0;
        int button_in3 = 0;
        int coins = 0;

        //Set LD1 through LD4 as digital output
        DeviceInit();
        //Initialize timer for delay
        DelayInit();
        //Initialize display
        DisplayInit(coins);


        /* Perform the main application loop*/
        while (1)
        {
                // Read buttons
                button_in12 = PORTReadBits (IOPORT_G, BIT_6|BIT_7);
                button_in3  = PORTReadBits (IOPORT_A, BIT_0);

                if (button_in12 != 0)
                {
                        // drive both LD1 and LD2 high if both buttons pressed
                        if (((button_in12 & 0x0040) != 0) &&
                            ((button_in12 & 0x0080) != 0))
                                coins = coins+15;
```

```
                        else
                        {
                                //drive LD1 high if only BTN1 pressed
                                if ((button_in12 & 0x0040) !=0) // BTN1 pressed?
                                        coins = coins+5;
                                //drive LD2 high if only BTN2 pressed
                                if ((button_in12 & 0x0080) != 0) // BTN2 pressed
                                        coins = coins+10;
                        }
                        DelayMs(1);
                }
                // Handle BTN3 separately
                if(button_in3 !=0)
                        coins=0;
                DelayMs(1);
        }
}

/* ------------------------------------------------------------------------ */
/*    DisplayInit()
**
**    Parameters:
**        coins             -amount of money entered
**        delay             -delay between blinks
**
**    Return Value:
**        none
**
**    Errors:
**        none
**
**    Description:
**        Set display state based on amount of money entered
/* ------------------------------------------------------------------------ */

void DisplayInit(int coins)
{
        int msdelay = 600;
        //Clear LEDs
        PORTClearBits(IOPORT_G, BIT_12|BIT_13|BIT_14|BIT_15);

        switch (coins)
        {
        case  5: PORTWrite (IOPORT_G, BIT_12);         //001
                break;
        case 10: PORTWrite (IOPORT_G, BIT_13);         //010
                break;
        case 15: PORTWrite (IOPORT_G, BIT_12|BIT_13);  //011
                break;
        case 20: PORTWrite (IOPORT_G, BIT_14);         //100
                break;
        case 25: PORTWrite (IOPORT_G, BIT_12|BIT_14);  //101
                break;
        case 30: PORTWrite (IOPORT_G, BIT_15);         //111
                DelayMs(msdelay);
                PORTClearBits(IOPORT_G, BIT_12|BIT_13|BIT_14|BIT_15);
                break;
        case 35: PORTWrite (IOPORT_G, BIT_12|BIT_13|BIT_14|BIT_15);  //111+
                DelayMs(msdelay);
                PORTClearBits(IOPORT_G, BIT_12|BIT_13|BIT_14);
                DelayMs(msdelay);
                PORTWrite (IOPORT_G, BIT_12|BIT_13|BIT_14);
                DelayMs(msdelay);
                PORTClearBits(IOPORT_G, BIT_12|BIT_13|BIT_14);
                DelayMs(msdelay);
                PORTWrite (IOPORT_G, BIT_12|BIT_13|BIT_14);
                DelayMs(msdelay);
                PORTClearBits(IOPORT_G, BIT_12|BIT_13|BIT_14|BIT_15);
                break;
        default: PORTClearBits(IOPORT_G, BIT_12|BIT_13|BIT_14|BIT_15);
        }
}

/* ------------------------------------------------------------------------ */
```

```
/*    DeviceInit()
**
**    Parameters:
**        none
**
**    Return Value:
**        none
**
**    Errors:
**        none
**
**    Description:
**        Set LD1 through LD4 as digital output
   ----------------------------------------------------------------------- */

void DeviceInit()
{
        // On MX7 board, disable JTAG function
        DDPCONbits.JTAGEN = 0;

        //On MX7 LED1 is on RG12
        //       LED2 is on RG13
        //       LED3 is on RG14
        //       LED4 is on RG15
        //Set ports for onboard LEDs to outputs & clear them
        PORTSetPinsDigitalOut (IOPORT_G, BIT_12|BIT_13| BIT_14|BIT_15);
        PORTClearBits(IOPORT_G, BIT_12|BIT_13| BIT_14|BIT_15);
        //Set ports for onboard BTNs as inputs
        PORTSetPinsDigitalIn (IOPORT_G, BIT_6 | BIT_7);
        PORTSetPinsDigitalIn (IOPORT_A, BIT_0);
}

/* ----------------------------------------------------------------------- */
/*    DelayInit
**
**    Parameters:
**        none
**
**    Return Value:
**        none
**
**    Errors:
**        none
**
**    Description:
**        Initialized the hardware for use by delay functions. This
**        initializes Timer 1 to count at 10Mhz.
/* ----------------------------------------------------------------------- */

void DelayInit()
{
        unsigned int tcfg;

        /* Configure Timer 1 to count a 10MHz with a period of 0xFFFF*/
        tcfg =
            T1_ON|T1_IDLE_CON|T1_SOURCE_INT|T1_PS_1_1|T1_GATE_OFF|T1_SYNC_EXT_OFF;
        OpenTimer1(tcfg, 0xFFFF);
}

/* ----------------------------------------------------------------------- */
/*    DelayMs
**
**    Parameters:
**        cms             - number of milliseconds to delay
**
**    Return Value:
**        none
**
**    Errors:
**        none
**
**    Description:
**        Delay the requested number of milliseconds. Uses Timer1.
/* ----------------------------------------------------------------------- */
```

```
241 void DelayMs(int cms)
    {
243         int ims;

245         for (ims=0; ims<cms; ims++)
            {
247                 WriteTimer1(0);      // reset timer
                    while (ReadTimer1() < cntMsDelay); // wait for interval of 1 mS
249         }

251 }
```

kirby_lab03.c

Conclusion

Laboratory 2 was designed to familiarize students with stack operations and to give us more practice with MIPS assembly language by learning to implement nested routines. This was critical to understanding how to properly use push and pop to store and retrieve data as necessary. We also again made use of various registers and the HI and LO portions of the multiplication function to detect overflows.